# CIS 500 — Software Foundations

## Midterm II

## Answer key

**November 16, 2005**

## Simply typed lambda-calculus

*The following questions refer to the simply typed lambda-calculus with booleans and error handling. The syntax, typing, and evaluation rules for this system are given on page 1 of the companion handout.*

1. (10 points) Write down the types of each of the following terms. If a term can be given many types, you should write down the *smallest* one. If a term does not type check, write NONE. Note: Recall that $\texttt{T} \to \texttt{T} \to \texttt{T}$ is parsed as $\texttt{T} \to (\texttt{T} \to \texttt{T})$.

   (a) $\lambda$`x:Bool`$\to$`Bool. x (x (x (x (x true))))`

   Type: ______________________________

   *Answer:* `(Bool`$\to$`Bool)` $\to$ `Bool`

   (b) `(`$\lambda$`x:Bool.` $\lambda$`y:Bool`$\to$`Bool`$\to$`Bool. true) false (`$\lambda$`z:Bool`$\to$`Bool. true)`

   Type: ______________________________

   *Answer: NONE. The function's second argument is not the correct type.*

   (c) `(`$\lambda$`x:Bool.` $\lambda$`y:Bool. error) false false false false false`

   Type: ______________________________

   *Answer:* `Bool`

   (d) $\lambda$`x:Bool`$\to$`Bool`$\to$`Bool. x (x error)`

   Type: ______________________________

   *Answer: NONE.* `x error` *must have type* `Bool`$\to$`Bool` *and cannot be supplied to* `x`*.*

   (e) `try (if (`$\lambda$`x:Bool. x) error then (error false) else error) with` $\lambda$`y:Bool`$\to$`Bool. y`

   Type: ______________________________

   *Answer:* `(Bool`$\to$`Bool)`$\to$`(Bool`$\to$`Bool)`

   *Grading scheme: Binary. Two points per answer.*

## References

*The following questions refer to the simply typed lambda-calculus with references. The syntax, typing, and evaluation rules for this system are given on page 3 of the companion handout.*

2. (10 points) Which of the following functions *could* evaluate to 42 when applied to a *single* argument and evaluated with a store of the appropriate type? Circle YES and give the argument and store if that is the case, and circle NO otherwise.

   For example, the term

   $$\lambda \texttt{x:Ref Nat. !x + 1}$$

   evaluates to 42 with argument $\texttt{l}_1$ and store $(\texttt{l}_1 \mapsto \texttt{41})$.

   (a) $\lambda$`x:Ref Nat. x`

   YES, with argument ______________________ and store ______________________________

   NO

   *Answer: No, this function returns a value that is a reference*

   (b) $\lambda$`x:Ref Nat. (x:= 3;` $\texttt{l}_1$ `:= 42; !x)`

   YES, with argument ______________________ and store ______________________________

   NO

   *Answer: YES, with argument* $\texttt{l}_1$ *and store* $(\texttt{l}_1 \mapsto \texttt{0})$

   (c) $\lambda$`f:Unit` $\to$ `Unit. (`$\texttt{l}_1$ `:= 3; f unit; !`$\texttt{l}_1$`)`

   YES, with argument ______________________ and store ______________________________

   NO

   *Answer: YES, with argument* $(\lambda \texttt{u:Unit. } \texttt{l}_1 \texttt{ := 42})$ *and store* $(\texttt{l}_1 \mapsto \texttt{0})$

3. (10 points) Suppose we add an increment operator (`t++`) to the simply typed lambda-calculus with references. This operator should increase the value stored in a numerical reference by one. For example, the result of evaluating the following term

`let x = ref 3 in ( x++ ; !x )`

with the empty store is the value `4`.

We start formalizing this idea by adding a new term form for the increment operator:

`t ++`

and a new computation rule for this new term form.

$$\frac{\mu(l) = \texttt{nv}}{\texttt{l++} \mid \mu \longrightarrow \texttt{unit} \mid [\texttt{l} \mapsto \texttt{succ nv}]\mu} \text{ L-INCRLOC}$$

(a) What congruence rule(s) should we add?

*Answer:*

$$\frac{t_1 \mid \mu \longrightarrow t_1' \mid \mu'}{t_1 \texttt{ ++} \mid \mu \longrightarrow t_1' \texttt{ ++} \mid \mu'} \text{ E-INCR}$$

(b) What typing rule(s) should we add?

*Answer:*

$$\frac{\Gamma \mid \Sigma \vdash t : \mathit{Ref\ Nat}}{\Gamma \mid \Sigma \vdash t \texttt{ ++} : \mathit{Unit}} \text{ T-INCR}$$

### Implementing a type checker

Consider an implementation of a type checker for the simply-typed lambda-calculus extended with sums. The syntax of types will be extended in the following way:

```
type ty =
  ...
  TySum of ty * ty
```

The syntax of terms will be extended in the following way:

```
type term =
  ...
  TmCase of info * term * (string * term) * (string * term)
  TmInl of info * term * ty
  TmInr of info * term * ty
```

Your job is to finish the implementation of the function

```
typeof : context → term → ty
```

This recursive function returns the type of a term in a particular context. The general form of the function is a pattern match on the the form of the term.

```
let rec typeof (ctx:context) (t:term) :ty =
  match t with
     ...          (* Branches for variables, abstractions, applications *)
    | TmInl(info, t0, tyAnnot) → ... (* Branch for inl *)
    | TmInr(info, t0, tyAnnot) →  ...(* Branch for inr *)
    | TmCase(info, t0, (x1, t1), (x2, t2)) →  ... (* Branch for case *)
```

In this implementation, a `context` is composed of variable bindings and has the following interface:

```
type binding = VarBind of ty
val emptycontext : context
val addbinding   : context → string → binding → context
val getbinding   : info → context → int → binding
```

4. (5 points) Recall the typing rule for `inl` expressions:

$$\frac{\Gamma \vdash \mathtt{t_1} : \mathtt{T_1}}{\Gamma \vdash \mathtt{inl}\ \mathtt{t_1}\ \mathtt{as}\ \mathtt{T_1}\mathtt{+}\mathtt{T_2} : \mathtt{T_1}\mathtt{+}\mathtt{T_2}} \qquad \text{(T-INL)}$$

One of the following segments of OCaml code correctly implements this rule. Circle the letter associated with the correct answer. The important differences are underlined.

(a)
```
TmInl(fi, t1, tyAnnot) →
    (match typeof ctx t1 with
      TySum(ty1, ty2) →
        if tyAnnot = ty1 then ty1
        else error fi "Injected data does not have expected type"
    | _ → error fi "Annotation is not a sum type")
```

(b)
```
TmInl(fi, t1, tyAnnot) →
    (match tyAnnot with
      TySum(ty1, ty2) →
        if typeof ctx t1 = ty1 then tyAnnot
        else error fi "Injected data does not have expected type"
    | _ → error fi "Annotation is not a sum type")
```

(c)
```
TmInl(fi, t1, tyAnnot) →
    (match typeof ctx t1 with
      TySum(ty1, ty2) →
        if tyAnnot = ty1 then tyAnnot
        else error fi "Injected data does not have expected type"
    | _ → error fi "Annotation is not a sum type")
```

(d)
```
TmInl(fi, t1, tyAnnot) →
    (match tyAnnot with
      TySum(ty1, ty2) →
        if ty1 = ty2 then typeof ctx t1
        else error fi "Injected data does not have expected type"
    | _ → error fi "Annotation is not a sum type")
```

*Answer: b*

*Grading scheme: Binary.*

5. (5 points) Recall the typing rule for case expressions:

$$\frac{\Gamma \vdash \mathtt{t}_0 : \mathtt{T}_1\mathtt{+T}_2 \qquad \Gamma, \mathtt{x}_1{:}\mathtt{T}_1 \vdash \mathtt{t}_1 : \mathtt{T} \qquad \Gamma, \mathtt{x}_2{:}\mathtt{T}_2 \vdash \mathtt{t}_2 : \mathtt{T}}{\Gamma \vdash \mathtt{case}\ \mathtt{t}_0\ \mathtt{of}\ \mathtt{inl}\ \mathtt{x}_1 \Rightarrow \mathtt{t}_1\ \mid\ \mathtt{inr}\ \mathtt{x}_2 \Rightarrow \mathtt{t}_2 : \mathtt{T}} \qquad \text{(T-CASE)}$$

One of the following segments of OCaml code correctly implements this rule. Circle the letter associated with the correct answer. The important differences are underlined.

(a)
```
TmCase(fi, t0, (x1, t1), (x2, t2)) →
  (match (typeof ctx t0) with
    TySum(ty1, ty2) →
      let tyLcase = typeof ctx t1 in
      let tyRcase = typeof ctx t2 in
      if tyLcase = tyRcase then tyLcase
      else error fi "Branches of case have different types"
  | _ → error fi "Expected sum type")
```

(b)
```
TmCase(fi, t0, (x1, t1), (x2, t2)) →
  (match (typeof ctx t0) with
    TySum(ty1, ty2) →
      let ctx' = addbinding ctx x1 (VarBind(typeof ctx t1)) in
      let ctx'' = addbinding ctx' x2 (VarBind(typeof ctx t2)) in
      let tyLcase = typeof ctx'' t1 in
      let tyRcase = typeof ctx'' t2 in
      if tyLcase = tyRcase then tyLcase
      else error fi "Branches of case have different types"
  | _ → error fi "Expected sum type")
```

(c)
```
TmCase(fi, t0, (x1, t1), (x2, t2)) →
  (match (typeof ctx t0) with
    TySum(ty1, ty2) →
      let ctx' = addbinding ctx x1 (VarBind(ty1)) in
      let ctx'' = addbinding ctx' x2 (VarBind(ty2)) in
      let tyLcase = typeof ctx'' t1 in
      let tyRcase = typeof ctx'' t2 in
      if tyLcase = tyRcase then tyLcase
      else error fi "Branches of case have different types"
  | _ → error fi "Expected sum type")
```

(d)
```
TmCase(fi, t0, (x1, t1), (x2, t2)) →
  (match (typeof ctx t0) with
    TySum(ty1, ty2) →
      let ctx' = addbinding ctx x1 (VarBind(ty1)) in
      let ctx'' = addbinding ctx x2 (VarBind(ty2)) in
      let tyLcase = typeof ctx' t1 in
      let tyRcase = typeof ctx'' t2 in
      if tyLcase = tyRcase then tyLcase
      else error fi "Branches of case have different types"
  | _ → error fi "Expected sum type")
```

*Answer: d*

*Grading scheme: Binary.*

## Proving type soundness

*The following questions refer to the simply-typed $\lambda$-calculus with unit and fix. The syntax, typing, and evaluation rules for this system are given on page 6 of the companion handout.*

6. (10 points) **Theorem (Preservation): If** $\Gamma \vdash \mathtt{t} : \mathtt{T}$ **and** $\mathtt{t} \longrightarrow \mathtt{t}'$ **then** $\Gamma \vdash \mathtt{t}' : \mathtt{T}$**.**

   Consider a proof of this theorem by induction on the typing derivation, $\Gamma \vdash \mathtt{t} : \mathtt{T}$. Show the case that occurs when the derivation ends with the rule T-FIX. In your proof you may use any of the following lemmas: *canonical forms*, *substitution*, *weakening*, *permutation*, and *inversion of typing*. These lemmas are listed in the companion handout on page 7. Be explicit about each step of the proof, and do not include any irrelevant information.

   *Answer: Assume that the last rule of the typing derivation is* T-FIX*:*

   $$\frac{\Gamma \vdash t_1 : T \to T}{\Gamma \vdash \mathit{fix}\ t_1 : T}\ \text{T-FIX}$$

   *In this case,* $t = \mathit{fix}\ t_1$ *where* $\Gamma \vdash t_1 : T \to T$*. There are two subcases of this proof, depending on how the term evaluates.*

   (a) $\mathit{fix}\ t_1 \longrightarrow \mathit{fix}\ t_1'$ *by rule* E-FIX*, where* $t_1 \longrightarrow t_1'$.
   *By induction, as* $\Gamma \vdash t_1 : T \to T$ *and* $t_1 \longrightarrow t_1'$*, we know that* $\Gamma \vdash t_1' : T \to T$.
   *By rule* T-FIX*, we know that* $\Gamma \vdash \mathit{fix}\, t_1' : T$.

   (b) $t_1$ *is a value* $(\lambda x{:}T_1.\ t_1')$ *and* $\mathit{fix}\ (\lambda x{:}T_1.\ t_1') \longrightarrow [x \mapsto \mathit{fix}\ t_1]\, t_1'$ *by rule* E-FIXBETA.
   *By inversion of typing, we know that* $\Gamma, x : T_1 \vdash t_1' : T_1$ *and that* $T_1 = T$.
   *By the substitution theorem, we know that* $\Gamma \vdash [x \mapsto \mathit{fix}\ t_1]\, t_1' : T$*, which is what we wanted to show.*

## Properties of Typed Languages

*The following questions refer to the simply typed $\lambda$-calculus with products and* `Bool`*. The syntax, typing, and evaluation rules for this system are given on page 8 of the companion handout.*

7. (20 points) Recall the following theorems about the simply typed $\lambda$-calculus with products and `Bool`:

   - **Progress**: If $\vdash \texttt{t} : \texttt{T}$, then either `t` is a value or else $\texttt{t} \longrightarrow \texttt{t}'$ for some $\texttt{t}'$.
   - **Preservation**: If $\Gamma \vdash \texttt{t} : \texttt{T}$ and $\texttt{t} \longrightarrow \texttt{t}'$, then $\Gamma \vdash \texttt{t}' : \texttt{T}$.
   - **Uniqueness of types**: Each term `t` has at most one type, and if `t` has a type, then there is exactly one derivation of that typing.

   In each problem below ((a) through (c)), we add a single rule to this language. Consider these additions separately. For each theorem, circle whether the theorem remains true or if it becomes false. If a theorem becomes false, give a counterexample showing why.

   (a) $$\frac{}{\Gamma \vdash \{\texttt{t}_1\texttt{,}\ \texttt{t}_2\} : \texttt{Nat}}\ \text{T-PairNat}$$

   Progress: TRUE FALSE, because...

   *Answer: becomes false because* $\vdash$ `succ {0, 0} : Nat` *but* `succ {0, 0}` *does not step and is not a value*

   Preservation: TRUE FALSE, because...

   *Answer: remains true*

   Uniqueness of types: TRUE FALSE, because...

   *Answer: becomes false because* `{0, 0}` *can be assigned the types* `Nat` *and* $\texttt{Nat} \times \texttt{Nat}$

   *Grading scheme: Each subpart was worth 2 pts. 1 point off for a bad or confused counterexample.*

   (b) $$\frac{}{\texttt{pred}\ \{\texttt{t}_1\texttt{,}\ \texttt{t}_2\} \longrightarrow \texttt{t}_1}\ \text{E-PredPair}$$

   Progress: TRUE FALSE, because...

   *Answer: remains true*

   Preservation: TRUE FALSE, because...

   *Answer: remains true*

   Uniqueness of types: TRUE FALSE, because...

   *Answer: remains true*

   *Grading scheme: Each subpart was worth 2pts.*

   (c) $$\frac{\Gamma \vdash \texttt{t}_1 : \texttt{T}_1 \quad \Gamma \vdash \texttt{t}_2 : \texttt{T}_2}{\Gamma \vdash \texttt{fst}\ \{\texttt{t}_1\texttt{,}\ \texttt{t}_2\} : \texttt{Nat}}\ \text{T-FstNat}$$

   Progress: TRUE FALSE, because...

   *Answer: remains true*

   Preservation: TRUE FALSE, because...

   *Answer: becomes false since* $\vdash$ `fst {true, true} : Nat`, `fst {true, true}` $\longrightarrow$ `true`, *and* $\not\vdash$ `true : Nat`.

   Uniqueness of types: TRUE FALSE, because...

   *Answer: becomes false since* `fst {true, true}` *can be assigned the types* `Nat` *and* `Bool`

   *Grading scheme: Progress: 2 pts, binary. Preservation and uniqueness of types: 3 pts each. 1–2 pts off for bad or confused counterexamples.*

## Derived forms

*The following questions refer to the simply typed $\lambda$-calculus with products and* `Bool`*. The syntax, typing, and evaluation rules for this system are given on page 8 of the companion handout.*

8. (10 points) Let $\lambda^I$ be the simply typed $\lambda$-calculus with products and `Bool`. We extend $\lambda^I$ to a language that we call $\lambda^E$ which has a new "`and`" construct as follows.

*New syntax:*

$$\begin{array}{lcll} \mathtt{t} & ::= & \ldots & \text{terms:} \\ & & \mathtt{and\ t\ t} & \text{conjunction} \end{array}$$

*New typing rules:*

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{Bool} \quad \Gamma \vdash \mathtt{t}_2 : \mathtt{Bool}}{\Gamma \vdash \mathtt{and}\ \mathtt{t}_1\ \mathtt{t}_2 : \mathtt{Bool}} \text{ T-AND}$$

*New evaluation rules:*

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{and}\ \mathtt{t}_1\ \mathtt{t}_2 \longrightarrow \mathtt{and}\ \mathtt{t}_1'\ \mathtt{t}_2} \text{ E-AND1} \qquad \frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\mathtt{and}\ \mathtt{v}_1\ \mathtt{t}_2 \longrightarrow \mathtt{and}\ \mathtt{v}_1\ \mathtt{t}_2'} \text{ E-AND2}$$

$$\frac{}{\mathtt{and\ true\ true} \longrightarrow \mathtt{true}} \text{ E-ANDTRUE}$$

$$\frac{}{\mathtt{and\ false}\ \mathtt{v}_2 \longrightarrow \mathtt{false}} \text{ E-ANDFALSE1} \qquad \frac{}{\mathtt{and}\ \mathtt{v}_1\ \mathtt{false} \longrightarrow \mathtt{false}} \text{ E-ANDFALSE2}$$

Rather than treat `and` as a primitive, we can try to make it a derived form by giving the following function $e$ from $\lambda^E$ to $\lambda^I$:

$$\begin{array}{rcl} e(\mathtt{x}) & = & \mathtt{x} \\ e(\lambda\mathtt{x}{:}\mathtt{T}.\ \mathtt{t}) & = & \lambda\mathtt{x}{:}\mathtt{T}.\ e(\mathtt{t}) \\ e(\mathtt{t}_1\ \mathtt{t}_2) & = & e(\mathtt{t}_1)\ e(\mathtt{t}_2) \\ e(\mathtt{true}) & = & \mathtt{true} \\ e(\mathtt{false}) & = & \mathtt{false} \\ e(\mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3) & = & \mathtt{if}\ e(\mathtt{t}_1)\ \mathtt{then}\ e(\mathtt{t}_2)\ \mathtt{else}\ e(\mathtt{t}_3) \\ e(\mathtt{t.1}) & = & (e(\mathtt{t})).\mathtt{1} \\ e(\mathtt{t.2}) & = & (e(\mathtt{t})).\mathtt{2} \\ e(\{\mathtt{t}_1, \mathtt{t}_2\}) & = & \{e(\mathtt{t}_1), e(\mathtt{t}_2)\} \\ e(\mathtt{and}\ \mathtt{t}_1\ \mathtt{t}_2) & = & \mathtt{if}\ e(\mathtt{t}_1)\ \mathtt{then}\ e(\mathtt{t}_2)\ \mathtt{else\ false} \end{array}$$

Are we successful? For each of the following properties, circle TRUE if it holds for this "derived form" and otherwise circle FALSE and give a counterexample.

(a) if $\mathtt{t} \rightarrow_E \mathtt{t}'$ then $e(\mathtt{t}) \rightarrow_I e(\mathtt{t}')$.

TRUE FALSE, because...

*Answer: False. A counterexample is*
*$t$ = `and true ((λx:Bool. x) true)` and $t'$ = `and true true`.*

*Grading scheme: 4 pts off for saying this remained true. 1–2 pts off for giving a bad or confused counterexample.*

(b) if $\Gamma \vdash^E \mathtt{t} : \mathtt{T}$ then $\Gamma \vdash^I e(\mathtt{t}) : \mathtt{T}$.

TRUE FALSE, because...

*Answer: True*

*Grading scheme: 3pts. Binary.*

(c) if $\mathtt{t}$ is a value then $e(\mathtt{t})$ is a value.

TRUE FALSE, because...

*Answer: True*

*Grading scheme: 3pts. Binary.*

# Companion handout

# Full definitions of the systems used in the exam

## Simply-typed lambda calculus with error handling and `Bool`

*Syntax*

| | | |
|---|---|---|
| $\mathtt{t}$ ::= | | *terms* |
| | $\mathtt{error}$ | *run-time error* |
| | $\mathtt{true}$ | *constant true* |
| | $\mathtt{false}$ | *constant false* |
| | $\mathtt{if\ t\ then\ t\ else\ t}$ | *conditional* |
| | $\mathtt{x}$ | *variable* |
| | $\lambda\mathtt{x{:}T.t}$ | *abstraction* |
| | $\mathtt{t\ t}$ | *application* |
| | $\mathtt{try\ t\ with\ t}$ | *trap errors* |
| $\mathtt{v}$ ::= | | *values* |
| | $\mathtt{true}$ | *true value* |
| | $\mathtt{false}$ | *false value* |
| | $\lambda\mathtt{x{:}T.t}$ | *abstraction value* |
| $\mathtt{T}$ ::= | | *types* |
| | $\mathtt{T}\to\mathtt{T}$ | *type of functions* |
| | $\mathtt{Bool}$ | *type of booleans* |
| $\Gamma$ ::= | | *type environments* |
| | $\emptyset$ | *empty type env.* |
| | $\Gamma,\mathtt{x{:}T}$ | *term variable binding* |

*Evaluation* $\boxed{\mathtt{t} \longrightarrow \mathtt{t}'}$

$$\mathtt{if\ true\ then\ t_2\ else\ t_3} \longrightarrow \mathtt{t_2} \qquad \text{(E-IfTrue)}$$

$$\mathtt{if\ false\ then\ t_2\ else\ t_3} \longrightarrow \mathtt{t_3} \qquad \text{(E-IfFalse)}$$

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{if\ t_1\ then\ t_2\ else\ t_3} \longrightarrow \mathtt{if\ t_1'\ then\ t_2\ else\ t_3}} \qquad \text{(E-If)}$$

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{t_1\ t_2} \longrightarrow \mathtt{t_1'\ t_2}} \qquad \text{(E-App1)}$$

$$\frac{\mathtt{t_2} \longrightarrow \mathtt{t_2'}}{\mathtt{v_1\ t_2} \longrightarrow \mathtt{v_1\ t_2'}} \qquad \text{(E-App2)}$$

$$(\lambda\mathtt{x{:}T_{11}.t_{12}})\ \mathtt{v_2} \longrightarrow [\mathtt{x} \mapsto \mathtt{v_2}]\mathtt{t_{12}} \qquad \text{(E-AppAbs)}$$

$$\mathtt{try\ v_1\ with\ t_2} \longrightarrow \mathtt{v_1} \qquad \text{(E-TryV)}$$

$$\mathtt{try\ error\ with\ t_2} \longrightarrow \mathtt{t_2} \qquad \text{(E-TryError)}$$

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{try\ t_1\ with\ t_2} \longrightarrow \mathtt{try\ t_1'\ with\ t_2}} \qquad \text{(E-Try)}$$

$$\mathtt{if\ error\ then\ t_2\ else\ t_3} \longrightarrow \mathtt{error} \qquad \text{(E-IfErr)}$$

$$\mathtt{error\ t_2} \longrightarrow \mathtt{error} \qquad \text{(E-AppErr1)}$$

$$\mathtt{v_1\ error} \longrightarrow \mathtt{error} \qquad \text{(E-AppErr2)}$$

*Typing* $\boxed{\Gamma \vdash \mathtt{t} : \mathtt{T}}$

$$\frac{\mathtt{x}{:}\mathtt{T} \in \Gamma}{\Gamma \vdash \mathtt{x} : \mathtt{T}} \qquad \text{(T-VAR)}$$

$$\frac{\Gamma, \mathtt{x}{:}\mathtt{T}_1 \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \vdash \lambda \mathtt{x}{:}\mathtt{T}_1.\mathtt{t}_2 : \mathtt{T}_1 \to \mathtt{T}_2} \qquad \text{(T-ABS)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \to \mathtt{T}_{12} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11}}{\Gamma \vdash \mathtt{t}_1\ \mathtt{t}_2 : \mathtt{T}_{12}} \qquad \text{(T-APP)}$$

$$\Gamma \vdash \mathtt{true} : \mathtt{Bool} \qquad \text{(T-TRUE)}$$

$$\Gamma \vdash \mathtt{false} : \mathtt{Bool} \qquad \text{(T-FALSE)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{Bool} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T} \qquad \Gamma \vdash \mathtt{t}_3 : \mathtt{T}}{\Gamma \vdash \mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 : \mathtt{T}} \qquad \text{(T-IF)}$$

$$\Gamma \vdash \mathtt{error} : \mathtt{T} \qquad \text{(T-ERROR)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}}{\Gamma \vdash \mathtt{try}\ \mathtt{t}_1\ \mathtt{with}\ \mathtt{t}_2 : \mathtt{T}} \qquad \text{(T-TRY)}$$

## Simply-typed lambda calculus with references (and `Unit`, `Nat`, `Bool`)

*Syntax*

| | | | |
|---|---|---|---|
| `t` | ::= | | *terms* |
| | | `unit` | *constant* `unit` |
| | | `x` | *variable* |
| | | $\lambda$`x:T.t` | *abstraction* |
| | | `t t` | *application* |
| | | `ref t` | *reference creation* |
| | | `!t` | *dereference* |
| | | `t:=t` | *assignment* |
| | | $l$ | *store location* |
| | | `true` | *constant true* |
| | | `false` | *constant false* |
| | | `if t then t else t` | *conditional* |
| | | `0` | *constant zero* |
| | | `succ t` | *successor* |
| | | `pred t` | *predecessor* |
| | | `iszero t` | *zero test* |
| `v` | ::= | | *values* |
| | | `unit` | *constant* `unit` |
| | | $\lambda$`x:T.t` | *abstraction value* |
| | | $l$ | *store location* |
| | | `true` | *true value* |
| | | `false` | *false value* |
| | | `nv` | *numeric value* |
| `T` | ::= | | *types* |
| | | `Unit` | *unit type* |
| | | `T`$\rightarrow$`T` | *type of functions* |
| | | `Ref T` | *type of reference cells* |
| | | `Bool` | *type of booleans* |
| | | `Nat` | *type of natural numbers* |
| $\mu$ | ::= | | *stores* |
| | | $\emptyset$ | *empty store* |
| | | $\mu, l =$ `v` | *location binding* |
| $\Gamma$ | ::= | | *type environments* |
| | | $\emptyset$ | *empty type env.* |
| | | $\Gamma,$ `x:T` | *term variable binding* |
| $\Sigma$ | ::= | | *store typings* |
| | | $\emptyset$ | *empty store typing* |
| | | $\Sigma, l$`:T` | *location typing* |
| `nv` | ::= | | *numeric values* |

| | |
|---|---|
| $\texttt{0}$ | *zero value* |
| $\texttt{succ nv}$ | *successor value* |

*Evaluation* $\boxed{\texttt{t} \mid \mu \longrightarrow \texttt{t}' \mid \mu'}$

$$\frac{\texttt{t}_1 \mid \mu \longrightarrow \texttt{t}_1' \mid \mu'}{\texttt{t}_1\ \texttt{t}_2 \mid \mu \longrightarrow \texttt{t}_1'\ \texttt{t}_2 \mid \mu'} \qquad \text{(E-APP1)}$$

$$\frac{\texttt{t}_2 \mid \mu \longrightarrow \texttt{t}_2' \mid \mu'}{\texttt{v}_1\ \texttt{t}_2 \mid \mu \longrightarrow \texttt{v}_1\ \texttt{t}_2' \mid \mu'} \qquad \text{(E-APP2)}$$

$$(\lambda\texttt{x}\texttt{:}\texttt{T}_{11}\texttt{.}\texttt{t}_{12})\ \texttt{v}_2 \mid \mu \longrightarrow [\texttt{x} \mapsto \texttt{v}_2]\texttt{t}_{12} \mid \mu \qquad \text{(E-APPABS)}$$

$$\frac{l \notin dom(\mu)}{\texttt{ref}\ \texttt{v}_1 \mid \mu \longrightarrow l \mid (\mu, l \mapsto \texttt{v}_1)} \qquad \text{(E-REFV)}$$

$$\frac{\texttt{t}_1 \mid \mu \longrightarrow \texttt{t}_1' \mid \mu'}{\texttt{ref}\ \texttt{t}_1 \mid \mu \longrightarrow \texttt{ref}\ \texttt{t}_1' \mid \mu'} \qquad \text{(E-REF)}$$

$$\frac{\mu(l) = \texttt{v}}{\texttt{!}l \mid \mu \longrightarrow \texttt{v} \mid \mu} \qquad \text{(E-DEREFLOC)}$$

$$\frac{\texttt{t}_1 \mid \mu \longrightarrow \texttt{t}_1' \mid \mu'}{\texttt{!}\texttt{t}_1 \mid \mu \longrightarrow \texttt{!}\texttt{t}_1' \mid \mu'} \qquad \text{(E-DEREF)}$$

$$l\texttt{:=}\texttt{v}_2 \mid \mu \longrightarrow \texttt{unit} \mid [l \mapsto \texttt{v}_2]\mu \qquad \text{(E-ASSIGN)}$$

$$\frac{\texttt{t}_1 \mid \mu \longrightarrow \texttt{t}_1' \mid \mu'}{\texttt{t}_1\texttt{:=}\texttt{t}_2 \mid \mu \longrightarrow \texttt{t}_1'\texttt{:=}\texttt{t}_2 \mid \mu'} \qquad \text{(E-ASSIGN1)}$$

$$\frac{\texttt{t}_2 \mid \mu \longrightarrow \texttt{t}_2' \mid \mu'}{\texttt{v}_1\texttt{:=}\texttt{t}_2 \mid \mu \longrightarrow \texttt{v}_1\texttt{:=}\texttt{t}_2' \mid \mu'} \qquad \text{(E-ASSIGN2)}$$

$$\texttt{if true then}\ \texttt{t}_2\ \texttt{else}\ \texttt{t}_3 \mid \mu \longrightarrow \texttt{t}_2 \mid \mu \qquad \text{(E-IFTRUE)}$$

$$\texttt{if false then}\ \texttt{t}_2\ \texttt{else}\ \texttt{t}_3 \mid \mu \longrightarrow \texttt{t}_3 \mid \mu \qquad \text{(E-IFFALSE)}$$

$$\frac{\texttt{t}_1 \mid \mu \longrightarrow \texttt{t}_1' \mid \mu'}{\texttt{if}\ \texttt{t}_1\ \texttt{then}\ \texttt{t}_2\ \texttt{else}\ \texttt{t}_3 \mid \mu \longrightarrow \texttt{if}\ \texttt{t}_1'\ \texttt{then}\ \texttt{t}_2\ \texttt{else}\ \texttt{t}_3 \mid \mu'} \qquad \text{(E-IF)}$$

$$\frac{\texttt{t}_1 \mid \mu \longrightarrow \texttt{t}_1' \mid \mu'}{\texttt{succ}\ \texttt{t}_1 \mid \mu \longrightarrow \texttt{succ}\ \texttt{t}_1' \mid \mu'} \qquad \text{(E-SUCC)}$$

$$\texttt{pred 0} \mid \mu \longrightarrow \texttt{0} \mid \mu \qquad \text{(E-PREDZERO)}$$

$$\texttt{pred (succ}\ \texttt{nv}_1\texttt{)} \mid \mu \longrightarrow \texttt{nv}_1 \mid \mu \qquad \text{(E-PREDSUCC)}$$

$$\frac{\texttt{t}_1 \mid \mu \longrightarrow \texttt{t}_1' \mid \mu'}{\texttt{pred}\ \texttt{t}_1 \mid \mu \longrightarrow \texttt{pred}\ \texttt{t}_1' \mid \mu'} \qquad \text{(E-PRED)}$$

$$\texttt{iszero 0} \mid \mu \longrightarrow \texttt{true} \mid \mu \qquad \text{(E-ISZEROZERO)}$$

$$\texttt{iszero (succ}\ \texttt{nv}_1\texttt{)} \mid \mu \longrightarrow \texttt{false} \mid \mu \qquad \text{(E-ISZEROSUCC)}$$

$$\frac{\texttt{t}_1 \mid \mu \longrightarrow \texttt{t}_1' \mid \mu'}{\texttt{iszero}\ \texttt{t}_1 \mid \mu \longrightarrow \texttt{iszero}\ \texttt{t}_1' \mid \mu'} \qquad \text{(E-ISZERO)}$$

*Typing* $\boxed{\Gamma \mid \Sigma \vdash \mathtt{t} : \mathtt{T}}$

$$\Gamma \mid \Sigma \vdash \mathtt{unit} : \mathtt{Unit} \qquad \text{(T-UNIT)}$$

$$\frac{\mathtt{x}{:}\mathtt{T} \in \Gamma}{\Gamma \mid \Sigma \vdash \mathtt{x} : \mathtt{T}} \qquad \text{(T-VAR)}$$

$$\frac{\Gamma, \mathtt{x}{:}\mathtt{T}_1 \mid \Sigma \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \mid \Sigma \vdash \lambda \mathtt{x}{:}\mathtt{T}_1.\mathtt{t}_2 : \mathtt{T}_1 \to \mathtt{T}_2} \qquad \text{(T-ABS)}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \to \mathtt{T}_{12} \qquad \Gamma \mid \Sigma \vdash \mathtt{t}_2 : \mathtt{T}_{11}}{\Gamma \mid \Sigma \vdash \mathtt{t}_1\ \mathtt{t}_2 : \mathtt{T}_{12}} \qquad \text{(T-APP)}$$

$$\frac{\Sigma(l) = \mathtt{T}_1}{\Gamma \mid \Sigma \vdash l : \mathtt{Ref}\ \mathtt{T}_1} \qquad \text{(T-LOC)}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t}_1 : \mathtt{T}_1}{\Gamma \mid \Sigma \vdash \mathtt{ref}\ \mathtt{t}_1 : \mathtt{Ref}\ \mathtt{T}_1} \qquad \text{(T-REF)}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t}_1 : \mathtt{Ref}\ \mathtt{T}_{11}}{\Gamma \mid \Sigma \vdash\ !\mathtt{t}_1 : \mathtt{T}_{11}} \qquad \text{(T-DEREF)}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t}_1 : \mathtt{Ref}\ \mathtt{T}_{11} \qquad \Gamma \mid \Sigma \vdash \mathtt{t}_2 : \mathtt{T}_{11}}{\Gamma \mid \Sigma \vdash \mathtt{t}_1 \mathtt{:=} \mathtt{t}_2 : \mathtt{Unit}} \qquad \text{(T-ASSIGN)}$$

$$\Gamma \mid \Sigma \vdash \mathtt{true} : \mathtt{Bool} \qquad \text{(T-TRUE)}$$

$$\Gamma \mid \Sigma \vdash \mathtt{false} : \mathtt{Bool} \qquad \text{(T-FALSE)}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t}_1 : \mathtt{Bool} \qquad \Gamma \mid \Sigma \vdash \mathtt{t}_2 : \mathtt{T} \qquad \Gamma \mid \Sigma \vdash \mathtt{t}_3 : \mathtt{T}}{\Gamma \mid \Sigma \vdash \mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 : \mathtt{T}} \qquad \text{(T-IF)}$$

$$\Gamma \mid \Sigma \vdash \mathtt{0} : \mathtt{Nat} \qquad \text{(T-ZERO)}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t}_1 : \mathtt{Nat}}{\Gamma \mid \Sigma \vdash \mathtt{succ}\ \mathtt{t}_1 : \mathtt{Nat}} \qquad \text{(T-SUCC)}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t}_1 : \mathtt{Nat}}{\Gamma \mid \Sigma \vdash \mathtt{pred}\ \mathtt{t}_1 : \mathtt{Nat}} \qquad \text{(T-PRED)}$$

$$\frac{\Gamma \mid \Sigma \vdash \mathtt{t}_1 : \mathtt{Nat}}{\Gamma \mid \Sigma \vdash \mathtt{iszero}\ \mathtt{t}_1 : \mathtt{Bool}} \qquad \text{(T-ISZERO)}$$

## Simply-typed lambda calculus with `Unit` and `fix`

*Syntax*

| | | |
|---|---|---|
| $\mathtt{t}$ ::= | | *terms* |
| | $\mathtt{x}$ | *variable* |
| | $\lambda\mathtt{x}{:}\mathtt{T}.\mathtt{t}$ | *abstraction* |
| | $\mathtt{t}\ \mathtt{t}$ | *application* |
| | $\mathtt{unit}$ | *constant unit* |
| | $\mathtt{fix}\ \mathtt{t}$ | *fix* |
| $\mathtt{v}$ ::= | | *values* |
| | $\lambda\mathtt{x}{:}\mathtt{T}.\mathtt{t}$ | *abstraction value* |
| | $\mathtt{unit}$ | *unit value* |
| $\mathtt{T}$ ::= | | *types* |
| | $\mathtt{T}\to\mathtt{T}$ | *type of functions* |
| | $\mathtt{Unit}$ | *type of unit* |
| $\Gamma$ ::= | | *contexts* |
| | $\emptyset$ | *empty context* |
| | $\Gamma, \mathtt{x}{:}\mathtt{T}$ | *term variable binding* |

*Evaluation* $\boxed{\mathtt{t} \longrightarrow \mathtt{t}'}$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\ \mathtt{t}_2 \longrightarrow \mathtt{t}_1'\ \mathtt{t}_2} \quad \text{(E-APP1)}$$

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\mathtt{v}_1\ \mathtt{t}_2 \longrightarrow \mathtt{v}_1\ \mathtt{t}_2'} \quad \text{(E-APP2)}$$

$$(\lambda\mathtt{x}{:}\mathtt{T}_{11}.\mathtt{t}_{12})\ \mathtt{v}_2 \longrightarrow [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12} \quad \text{(E-APPABS)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{fix}\ \mathtt{t}_1 \longrightarrow \mathtt{fix}\ \mathtt{t}_1'} \quad \text{(E-FIX)}$$

$$\mathtt{fix}\ (\lambda\mathtt{x}{:}\mathtt{T}_1.\mathtt{t}_2) \longrightarrow [\mathtt{x} \mapsto \mathtt{fix}\ (\lambda\mathtt{x}{:}\mathtt{T}_1.\mathtt{t}_2)]\mathtt{t}_2 \quad \text{(E-FIXBETA)}$$

*Typing* $\boxed{\Gamma \vdash \mathtt{t} : \mathtt{T}}$

$$\frac{\mathtt{x}{:}\mathtt{T} \in \Gamma}{\Gamma \vdash \mathtt{x} : \mathtt{T}} \quad \text{(T-VAR)}$$

$$\frac{\Gamma, \mathtt{x}{:}\mathtt{T}_1 \vdash \mathtt{t}_2 : \mathtt{T}_2}{\Gamma \vdash \lambda\mathtt{x}{:}\mathtt{T}_1.\mathtt{t}_2 : \mathtt{T}_1\to\mathtt{T}_2} \quad \text{(T-ABS)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11}\to\mathtt{T}_{12} \qquad \Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11}}{\Gamma \vdash \mathtt{t}_1\ \mathtt{t}_2 : \mathtt{T}_{12}} \quad \text{(T-APP)}$$

$$\frac{}{\Gamma \vdash \mathtt{unit} : \mathtt{Unit}} \quad \text{(T-UNIT)}$$

$$\frac{\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_1\to\mathtt{T}_1}{\Gamma \vdash \mathtt{fix}\ \mathtt{t}_1 : \mathtt{T}_1} \quad \text{(T-FIX)}$$

*Properties of STLC + $unit$ + $fix$*

1. *Lemma [Canonical forms]:*
   (a) If $\mathtt{v}$ is a value of type $\mathtt{T}_1 \to \mathtt{T}_2$, then $\mathtt{v}$ has the form $\lambda \mathtt{x}{:}\mathtt{T}_1.\mathtt{t}_2$.
   (b) If $\mathtt{v}$ is a value of type $\mathtt{Unit}$, then $\mathtt{v}$ is $\mathtt{unit}$.
2. *Lemma [Substitution]:* If $\Gamma, \mathtt{x}{:}\mathtt{S} \vdash \mathtt{t} : \mathtt{T}$ and $\Gamma \vdash \mathtt{s} : \mathtt{S}$, then $\Gamma \vdash [\mathtt{x} \mapsto \mathtt{s}]\mathtt{t} : \mathtt{T}$.
3. *Lemma [Permutation]:* If $\Gamma \vdash \mathtt{t} : \mathtt{T}$ and $\Delta$ is a permutation of $\Gamma$ then $\Delta \vdash \mathtt{t} : \mathtt{T}$. Moreover the latter derivation has the same depth as the former.
4. *Lemma [Weakening]:* If $\Gamma \vdash \mathtt{t} : \mathtt{T}$ and $\mathtt{x} \notin \mathit{dom}(\Gamma)$ then $\Gamma, \mathtt{x} : \mathtt{S} \vdash \mathtt{t} : \mathtt{T}$. Moreover the latter derivation has the same depth as the former.
5. *Lemma [Inversion of typing]:*
   (a) If $\Gamma \vdash \mathtt{x} : \mathtt{R}$, then $\mathtt{x}{:}\mathtt{R} \in \Gamma$.
   (b) If $\Gamma \vdash \lambda \mathtt{x}{:}\mathtt{T}_1.\mathtt{t}_2 : \mathtt{R}$, then $\mathtt{R} = \mathtt{T}_1 \to \mathtt{R}_2$ for some $\mathtt{R}_2$ with $\Gamma, \mathtt{x}{:}\mathtt{T}_1 \vdash \mathtt{t}_2 : \mathtt{R}_2$.
   (c) If $\Gamma \vdash \mathtt{t}_1\ \mathtt{t}_2 : \mathtt{R}$, then there is some type $\mathtt{T}_{11}$ such that $\Gamma \vdash \mathtt{t}_1 : \mathtt{T}_{11} \to \mathtt{R}$ and $\Gamma \vdash \mathtt{t}_2 : \mathtt{T}_{11}$.
   (d) If $\Gamma \vdash \mathtt{unit} : \mathtt{R}$, then $\mathtt{R} = \mathtt{Unit}$.
   (e) If $\Gamma \vdash \mathtt{fix}\,\mathtt{t}_1 : \mathtt{R}$, then $\Gamma \vdash \mathtt{t}_1 : \mathtt{R} \to \mathtt{R}$,

## Simply-typed lambda calculus with products and `Bool`

*Syntax*

| | | |
|---|---|---|
| t ::= | | *terms* |
| | $\mathtt{x}$ | *variable* |
| | $\lambda\mathtt{x}\!:\!\mathtt{T}.\mathtt{t}$ | *abstraction* |
| | $\mathtt{t}\ \mathtt{t}$ | *application* |
| | $\mathtt{true}$ | *constant true* |
| | $\mathtt{false}$ | *constant false* |
| | $\mathtt{if}\ \mathtt{t}\ \mathtt{then}\ \mathtt{t}\ \mathtt{else}\ \mathtt{t}$ | *conditional* |
| | $\{\mathtt{t},\mathtt{t}\}$ | *pair* |
| | $\mathtt{t}.1$ | *first projection* |
| | $\mathtt{t}.2$ | *second projection* |
| v ::= | | *values* |
| | $\lambda\mathtt{x}\!:\!\mathtt{T}.\mathtt{t}$ | *abstraction value* |
| | $\mathtt{true}$ | *true value* |
| | $\mathtt{false}$ | *false value* |
| | $\{\mathtt{v},\mathtt{v}\}$ | *pair value* |
| T ::= | | *types* |
| | $\mathtt{T}\to\mathtt{T}$ | *type of functions* |
| | $\mathtt{Bool}$ | *type of booleans* |
| | $\mathtt{T}_1 \times \mathtt{T}_2$ | *product type* |
| $\Gamma$ ::= | | *type environments* |
| | $\emptyset$ | *empty type env.* |
| | $\Gamma, \mathtt{x}\!:\!\mathtt{T}$ | *term variable binding* |

*Evaluation* $\boxed{\mathtt{t} \longrightarrow \mathtt{t}'}$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1\ \mathtt{t}_2 \longrightarrow \mathtt{t}_1'\ \mathtt{t}_2} \quad \text{(E-App1)}$$

$$\frac{\mathtt{t}_2 \longrightarrow \mathtt{t}_2'}{\mathtt{v}_1\ \mathtt{t}_2 \longrightarrow \mathtt{v}_1\ \mathtt{t}_2'} \quad \text{(E-App2)}$$

$$(\lambda\mathtt{x}\!:\!\mathtt{T}_{11}.\mathtt{t}_{12})\ \mathtt{v}_2 \longrightarrow [\mathtt{x} \mapsto \mathtt{v}_2]\mathtt{t}_{12} \quad \text{(E-AppAbs)}$$

$$\mathtt{if}\ \mathtt{true}\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 \longrightarrow \mathtt{t}_2 \quad \text{(E-IfTrue)}$$

$$\mathtt{if}\ \mathtt{false}\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 \longrightarrow \mathtt{t}_3 \quad \text{(E-IfFalse)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{if}\ \mathtt{t}_1\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3 \longrightarrow \mathtt{if}\ \mathtt{t}_1'\ \mathtt{then}\ \mathtt{t}_2\ \mathtt{else}\ \mathtt{t}_3} \quad \text{(E-If)}$$

$$\{\mathtt{v}_1,\mathtt{v}_2\}.1 \longrightarrow \mathtt{v}_1 \quad \text{(E-PairBeta1)}$$

$$\{\mathtt{v}_1,\mathtt{v}_2\}.2 \longrightarrow \mathtt{v}_2 \quad \text{(E-PairBeta2)}$$

$$\frac{\mathtt{t}_1 \longrightarrow \mathtt{t}_1'}{\mathtt{t}_1.1 \longrightarrow \mathtt{t}_1'.1} \quad \text{(E-Proj1)}$$

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{t_1.2} \longrightarrow \mathtt{t_1'.2}} \qquad \text{(E-Proj2)}$$

$$\frac{\mathtt{t_1} \longrightarrow \mathtt{t_1'}}{\mathtt{\{t_1,t_2\}} \longrightarrow \mathtt{\{t_1',t_2\}}} \qquad \text{(E-Pair1)}$$

$$\frac{\mathtt{t_2} \longrightarrow \mathtt{t_2'}}{\mathtt{\{v_1,t_2\}} \longrightarrow \mathtt{\{v_1,t_2'\}}} \qquad \text{(E-Pair2)}$$

*Typing* $\boxed{\Gamma \vdash \mathtt{t} : \mathtt{T}}$

$$\frac{\mathtt{x:T} \in \Gamma}{\Gamma \vdash \mathtt{x} : \mathtt{T}} \qquad \text{(T-Var)}$$

$$\frac{\Gamma, \mathtt{x:T_1} \vdash \mathtt{t_2} : \mathtt{T_2}}{\Gamma \vdash \lambda\mathtt{x:T_1.t_2} : \mathtt{T_1 \to T_2}} \qquad \text{(T-Abs)}$$

$$\frac{\Gamma \vdash \mathtt{t_1} : \mathtt{T_{11} \to T_{12}} \qquad \Gamma \vdash \mathtt{t_2} : \mathtt{T_{11}}}{\Gamma \vdash \mathtt{t_1\ t_2} : \mathtt{T_{12}}} \qquad \text{(T-App)}$$

$$\Gamma \vdash \mathtt{true} : \mathtt{Bool} \qquad \text{(T-True)}$$

$$\Gamma \vdash \mathtt{false} : \mathtt{Bool} \qquad \text{(T-False)}$$

$$\frac{\Gamma \vdash \mathtt{t_1} : \mathtt{Bool} \qquad \Gamma \vdash \mathtt{t_2} : \mathtt{T} \qquad \Gamma \vdash \mathtt{t_3} : \mathtt{T}}{\Gamma \vdash \mathtt{if}\ \mathtt{t_1}\ \mathtt{then}\ \mathtt{t_2}\ \mathtt{else}\ \mathtt{t_3} : \mathtt{T}} \qquad \text{(T-If)}$$

$$\frac{\Gamma \vdash \mathtt{t_1} : \mathtt{T_1} \qquad \Gamma \vdash \mathtt{t_2} : \mathtt{T_2}}{\Gamma \vdash \mathtt{\{t_1,t_2\}} : \mathtt{T_1} \times \mathtt{T_2}} \qquad \text{(T-Pair)}$$

$$\frac{\Gamma \vdash \mathtt{t_1} : \mathtt{T_{11}} \times \mathtt{T_{12}}}{\Gamma \vdash \mathtt{t_1.1} : \mathtt{T_{11}}} \qquad \text{(T-Proj1)}$$

$$\frac{\Gamma \vdash \mathtt{t_1} : \mathtt{T_{11}} \times \mathtt{T_{12}}}{\Gamma \vdash \mathtt{t_1.2} : \mathtt{T_{12}}} \qquad \text{(T-Proj2)}$$